6 ALGORITMO

Nesse artigo, compararemos algumas estratégias diferentes para diminuir o tempo máximo de espera em uma linha de trens/ônibus. Tentaremos encontrar estratégias que aproveitem o fato do *Tarjan* ser linear.

Identificaremos os ciclos maximais aplicando o autovetor associado através do algoritmo de *Floria* e, por fim, precisaremos aplicar *Tarjan* para identificar quais nós de fato estão no ciclo maximal (alguns ramos não conexos aparecem ao aplicar autovetor).

Como *Tarjan* funciona através de uma Busca em Profundidade, teremos uma árvore de busca, e todos os ciclos podem ser identificados através dos *Nós de Retorno*. A fim de diminuir a média dos ciclos maximais, sempre que encontrarmos um ciclo, adicionaremos um nó a ele, dividindo os pesos em duas arestas. Isso, na prática, significa adicionar uma nova saída de ônibus a partir daquela linha, por exemplo, se as saídas, a partir de uma estação, eram a cada 30 minutos, agora elas serão a cada 15. A seguir um exemplo visual:

Perceba que inicialmente o ciclo tinha custo médio $c = \frac{30+7+5}{3} = 14$. Ao fim, porém, temos que o custo é $c_{final} = \frac{15+15+7+5}{4} = 10,5$. Chamaremos esse algoritmo de **Algoritmo Cego**, uma vez que ele adiciona nós a qualquer ciclo sem ponderar se essa é a melhor estratégia.

Podemos, por fim, aplicar o algoritmo iterativamente até que $c_{final} \leq T$ (T é o período desejado, escolhido arbitrariamente).

6.1 Descrição Formal (Algoritmo Açougue) - Cego

Seja $G_0 = (V, E, w)$ um grafo orientado ponderado, onde $w : E \to \mathbb{R}$ é uma função de peso. Seja $T \in \mathbb{R}$ um limiar de média esperada.

O objetivo do algoritmo é modificar iterativamente o grafo para que sua **Constante Cíclica Maximal**, denotada por $\lambda(G_0)$, seja reduzida a um valor menor ou igual a T.

O algoritmo opera sobre um grafo auxiliar $G' = (V, E, w')$, onde os pesos são invertidos, $w'_{ij} = -w_{ij}$ para toda aresta $(i, j) \in E$. A Constante Cíclica Minimal de G', está relacionada à Constante Maximal de G_0 pela identidade:

$$\lambda(G') = -\lambda(G_0)$$

1. **Inicialização ($k = 0$):**
   (a) Calcule a Constante inicial $\lambda_0 = \lambda(G_0)$ (via algoritmo de Floria-Grithhs REF).
   (b) Gere o grafo de pesos invertidos G'_0, com pesos $w'_{ij} = -w_{ij}$ (retornado por `floria` como **custo_invertido** REF).
   (c) Defina o grafo atual $H_0 \leftarrow G'_0$.
   (d) Defina a média atual $\mu_0 \leftarrow \lambda_0$.
2. **Iteração (para $k = 0, 1, 2, \ldots$):**
   (a) **Verificação de Parada:** Se $\mu_k \geq -T$ (equivalente a $\lambda^*(G_k) \leq T$) ou se o número

de iterações exceder 200, **terminar** e retornar o grafo H_k.

(b) **Construção do Grafo Crítico:**

(i) Calcule o autovetor v (retornado por floria como vetor_atual).

(ii) Construa o grafo crítico $C_k = (V_c, E_c)$ como um subgrafo de H_k (usando **grafoCritico_adj** REF)

(c) **Corte do Ciclo (Tarjan Modificado):**

(i) Execute o algoritmo **tarjan_subgrafo_lista** no grafo H_k, usando C_k como o subgrafo-guia.

(ii) Este algoritmo identifica todos os ciclos elementares presentes em C_k.

(iii) Para cada ciclo $\mathcal{C}$ encontrado, o algoritmo identifica a aresta (u, v) imediata de retorno.

(iv) A aresta (u, v) é "cortada"no grafo H_k: um novo nó z é adicionado e a aresta (u, v) é substituída por (u, z) e (z, v), cada uma com metade do peso original.

(v) Seja H_{k+1} o grafo resultante.

(d) **Recálculo:**

(i) Calcule a nova Constante lica Maximal $\mu_{k+1} = \lambda(H_{k+1})$.

(ii) Incremente $k \leftarrow k + 1$ e retorne ao passo 2(a).

6.2 Açougue Consciente

Perceba que o algoritmo da forma que foi implementado funciona. Porém, ele gera rapidamente arestas com pesos muito próximos de 0, uma vez que se um ciclo $\mathcal{C}$ necessitar de n iterações para estar sob o limiar T, o peso da aresta de retorno do DFS que aplicamos terá valor $c_n = \frac{c_0}{n}$. Portanto, caso $n \to \infty$, $c_n \to 0$.

Para corrigir isso, portanto, basta alterar o passo **2c (iii)**. Segue-se a correção:

2c(iii): *Para cada ciclo $\mathcal{C}$ detectado, o algoritmo encontra a aresta $(u, v) \in \mathcal{C}$ que maximiza o peso absoluto (usando w'):*

$$(u, v) = \arg\max_{(i,j)\in\mathcal{C}} |w'_{ij}|$$

Isso é feito armazenando os pais de cada nó visitado e percorrendo de trás para frente até v checando a aresta com maior peso.

6.3 Açougue Centros

A ideia desse algoritmo é reutilizar arestas criadas em iterações anteriores. Dessa forma, podemos encontrar a quantidade mínima de Nós ("Centros de Distribuição") que podemos adicionar a nossa malha a fim de chegarmos no nosso objetivo. Para isso, basta modificar o passo **2c(iv)** do nosso algoritmo, da seguinte forma:

2c(iv) *A aresta (u, v) é "cortada"no grafo H_k: checamos se não há z_i, tal que z_i foi criado em uma iteração anterior e não está em $\mathcal{C}_{\|}$. Caso haja, a aresta (u, v) é substituída por (u, z) e (z, v), cada uma com metade do peso original. Caso não haja, basta fazer o mesmo que o algoritmo anterior faria.*

Perceba que isso altera significativamente a topologia da nossa malha. Por vezes, conectaremos duas regiões que não serão conectadas. Podemos ainda, gerar novos ciclos, inclusive que podem ser Maximais. Perceba, porém, que essa abordagem é muito interessante para encontrarmos o valor mínimo de T, sob o qual só precisamos adicionar n saídas. Isto é, digamos que temos a possibilidade de adicionar mais uma saída em nossa malha, com esse algoritmo, podemos encontrar a melhor forma de adiciona-la de maneira que minimizaremos os ciclos maximais ("Gargalos") existentes.

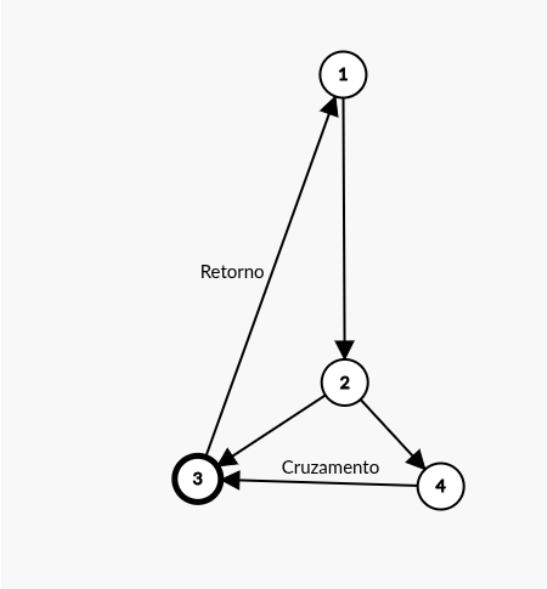

Figura 7 – Tipos de Nós

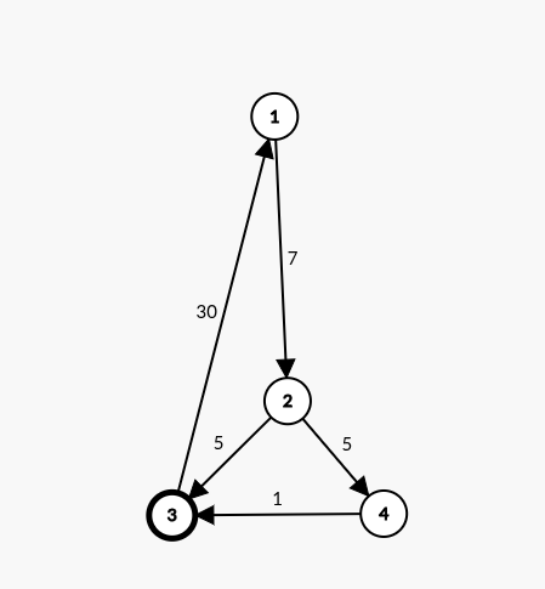

Figura 8 – Grafo ponderado

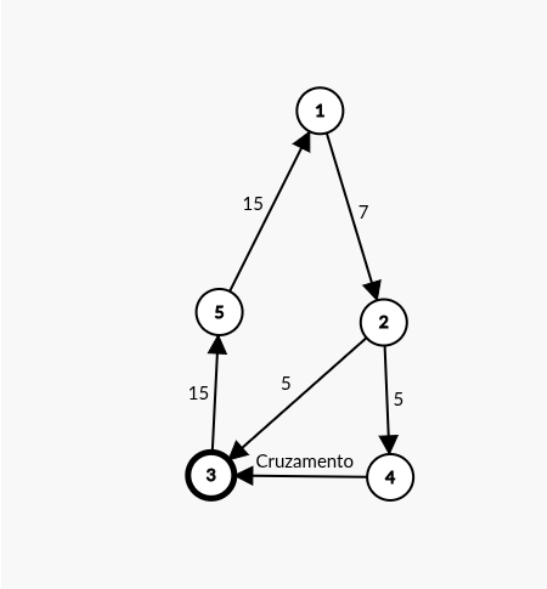

Figura 9 – Grafo Final